НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“___” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: Система побудови зображення 3D моделей з відстеженням зворотної траєкторії променя _____
_____

Виконав: студент IV курсу, групи КВ-53

_____Малишкін Юрій Олександрович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст.викладач каф.СПСКС, к.т.н. Наливайчук М.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент доцент каф. ОТ ФІОТ Корочкін О.В. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки **6.050102 «Комп'ютерна інженерія»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Тарасенко В.П.

(підпис) (ініціали, прізвище)

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проект студенту

Малишкіну Юрію Олександровичу

(прізвище, ім'я, по батькові)

1. Тема проекту "Система побудови зображення 3D моделей з відстеженням зворотної траєкторії променя",

керівник проекту ст.викладач каф. СП і СКС, к.т.н., Наливайчук М.В.,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «06»травня 2019 р. №1330-С

2. Термін подання студентом проекту 10.06.2019.

3. Вихідні дані до проекту

Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя

4. Зміст пояснювальної записки

Аналіз основних методів рендерингу зображень.

Техніка візуалізації відстеження променів.

Структура методу побудови зображення за допомогою зворотного відстеження променя.

Тестування програмних засобів.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

ІАЛЦ.045490.005 Д1. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема функціональна роботи системи.

ІАЛЦ.045490.006 Д2. Підпрограма побудови зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема алгоритму підпрограми.

ІАЛЦ.045490.007 Д3. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема структурна.

ІАЛЦ.045490.008 Д4. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема функціональна затінення.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Н.контроль	Клятченко Я.М., к.т.н., доцент		

7. Дата видачі завдання 16.10.2018

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка

1.	Видача завдання на дипломне проектування	13.02.2019	
2.	Розробка технічного завдання	15.04.2019	
3.	Аналіз існуючих рішень	05.05.2019	
4.	Вибір середовища розробки	10.05.2019	
5.	Розробка програмного продукту	18.05.2019	
6.	Відлагодження програмного продукту	20.05.2019	
7.	Підготовка пояснювальної записки	25.05.2019	
8.	Оформлення матеріалів проекту	27.05.2019	
9.	Попередній огляд матеріалів диплому на кафедрі	30.05.2019	

Студент _____ _____

(підпис) (ініціали, прізвище)

Керівник проекту _____ _____

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка має обсяг 53 сторінки, 28 ілюстрації, 15 бібліографічних посилань.

В дипломній роботі розглянуто питання створення системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя.

Трасування променів являє собою спосіб зображення 3D-об’єктів за допомогою спостереженням за поведінкою проходження світлового променя крізь точку екрану та імітації взаємодії цього променя з будь-якими об’єктами, що підлягають відображенню. За допомогою цього способу створюються надзвичайно реалістичні зображення, кращі, ніж можуть створити альтернативні способи.

Трасування променів здатне працювати з багатою кількістю оптичних опцій, таких як заломлення променів, їх відбиття, розсіювання або хроматичну аберацію.

Основною задачею трасування променів є проектування променя для одного пікселя в 3D-середовищі та з’ясування, який полігон потрапляє в перший промінь, а потім відповідним чином зафарбування його.

Розроблена система трасування променів у даній дипломній роботі забезпечить розрахунок різних властивостей оптичних шляхів.

Ключові слова: ray tracing, scanline, ray casting, rendering, computer graphics, 3D models, radiosity, z-buffer, rasterization, пряма трасування, відстеження назад, промінь тіні, трасування траєкторії.

ABSTRACT

The explanatory note has a volume of 53 pages, 28 illustrations, 15 bibliographic references.

In diploma work is considered the question of creating a system for constructing an image of 3D models with trace back beam trajectory.

Ray tracing is a way of representing 3D objects by observing the behavior of passing a light beam through the screen point and simulating the interaction of this beam with any objects to be displayed. With this method, you create extremely realistic images that are better than alternative ways of creating.

Ray tracing can handle a large number of optical options, such as ray refraction, reflection, scattering, or chromatic aberration.

The main task of ray tracing is to design a beam for one pixel in a 3D environment and find out which polygon falls into the first ray and then properly color it.

The developed system of tracing beams in this diploma work will provide calculation of different properties of optical paths.

Key words: ray tracing, scanline, ray casting, rendering, computer graphics, 3D models, radiosity, z-buffer, rasterization, path tracing, backward tracing, shadow ray, forward tracing.

АННОТАЦИЯ

В дипломной работе рассмотрены вопросы создания системы построения изображения 3D-моделей с с отслеживанием обратной траектории луча.

Трассировка лучей представляет собой способ изображения 3D-обьектов с помощью наблюдения за повидением прохождения светового луча через точку экрана и иммитации взаимодействия этого луча с любыми объектами, подлежащими отражению. С помощью этого способа создаются чрезвычайно реалистичные изображения, лучше, чем могут создать альтернативные способы.

Трассировка лучей способна работать с богатым количеством оптических опций, таких как преломление лучей, их отражение, рассеивание или хроматическую абберацию.

Основной задачей рассировки лучей является проектирование луча для одного пикселя в 3D-среде и выяснения, какой полигон попадает в первый луч, а затем соответствующим образом окраски его.

Разработанная система трассировки лучей в данной дипломной работе обеспечит расчет различных свойств оптических путей.

Ключевые слова: ray tracing, scanline, ray casting, rendering, computer graphics, 3D models, radiosity, z-buffer, rasterization, прямая трассировки, обратное отслеживание, промень тени, трассировка траектории.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.002 ТЗ	Система побудови зображення	4		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Технічне завдання			
	А4	ІАЛЦ. 045490.003 ТП	Система побудови зображення	3		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Система побудови зображення	52		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Пояснювальна записка.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.001 ОА		
Розробив		Малишкін Ю.О.			Система побудови зображення 3D моделей з відстеженням зворотної траєкторії променя Опис альбому	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Наливайчук М.В				КПІ ім. Ігоря Сікорського ФПМ КВ-53	
Н. контроль		Клятченко Я.М.					
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045490.005 Д1	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Система побудови зображення			
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Схема функціональна.			
	А4	ІАЛЦ.045490.006 Д2	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Алгоритм роботи			
			підпрограми.			
			Схема алгоритму.			
	А4	ІАЛЦ.045490.007 Д3	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Система побудови зображення			
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Схема структурна.			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.008 Д4	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Програмні засоби.			
			Система побудови зображення			
			3D моделей з використанням			
			затінення.			
			Схема функціональна.			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Система побудови зображеня 3D моделей з відстеженням зворотної траєкторії променя».

Галузь застосування: дослідження рендерингу за допомогою комп’ютерної програми.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи ступеня «бакалавр комп'ютерної інженерії», затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є відтворення реалістичних зображень за допомогою алгоритму трасування променів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Сумісність з операційною системою Windows або Linux;
- можливість запису зображення на диск;
- забезпечення трасування променів.

- здатність зображення працювати з освітленням, дзеркальним освітленням, тінями та заломленням.
- Забаезпечення простоти при використанні користувачем;

5.2. Вимоги до апаратного забезпечення

- Процесор: 4-ядерний Intel Core i3-8100;
- Оперативна пам’ять: 4 Гб;

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows 10 або Linux;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.04.2019
2.	Розроблення та узгодження технічного завдання	30.04.2019
3.	Аналіз існуючих рішень	05.05.2019
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2019
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2019
6.	Підготовка графічної частини дипломного проекту	20.05.2019
7.	Оформлення документації дипломного проекту	25.05.2019
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2019

Поз.	Формат	Позначення	Найменування	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ. 045490.004 ПЗ	Система побудови зображення	53		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Пояснювальна записка.			
	А4	ІАЛЦ. 045490.005 Д1	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Система побудови зображення			
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Схема функціональна.			
	А4	ІАЛЦ.045490.006 Д2	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Алгоритм роботи			
			підпрограми.			
			Схема алгоритму.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ВП		
Розробив		Малишкін Ю.О.			Система побудови зображення 3D моделей з відстеженням зворотної траєкторії променя Відомість технічного проекту	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Наливайчук М.В.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-53	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.007 Д3	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Система побудови зображення			
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Схема структурна.			
	А4	ІАЛЦ.045490.008 Д4	Система побудови зображення	1		
			3D моделей з відстеженням			
			зворотної траєкторії променя.			
			Програмні засоби.			
			Система побудови зображення			
			3D моделей з використанням			
			затінення.			
			Схема функціональна.			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			

Пояснювальна записка
до дипломного проекту

на тему: Система побудови зображення 3D моделей з відстеженням зворотної траєкторії променя

Київ – 2019 року

ЗМІСТ

Додатки

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045490.005 Д1. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема функціональна роботи системи.
- ІАЛЦ.045490.006 Д2. Підпрограма побудови зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема алгоритму підпрограми.
- ІАЛЦ.045490.007 Д3. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема структурна.
- ІАЛЦ.045490.007 Д4. Побудова зображення 3D моделей з відстеженням зворотної траєкторії променя. Схема алгоритму затінення.

Додаток 2. Лістинг програми

Додаток 3. Копія презентації

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

CG – Computer Graphics

RT – Ray Tracing

RC – Ray Casting

ВСТУП

На протязі останніх років багато фахівців досліджують галузь, яку називають комп'ютерною графікою, завдяки високим показникам прогресу комп'ютерних технологій. Переважний об'єм інформації людина отримує візуально або пов'язує її з геометричними просторовими поняттями. Комп'ютерна графіка має великий потенціал для зменшення важкості процесу зрозуміння і творчості.

Деякі теми в комп'ютерній графіці включають в себе дизайн інтерфейсу, графіку спрайтів, векторну графіку, 3D-моделювання, шейдери, дизайн графічного процесора, неявний рендеринг поверхні, який простежується променями, а також комп'ютерне бачення. Загальна методологія сильно залежить від основних наук геометрії, оптики та фізики.

Тезис "комп'ютерна графіка" зазвичай пояснюється по-різному. Деякі ресурси відносять комп'ютерну графіку як до області інформатики, яка працює з проблемами отримання різних зображень (малюнків, анімацій) на комп'ютерній машині. За думкою інших джерел, вважається, що комп'ютерна графіка персоніфікує нову область розуміння, яка, з одного боку, являє собою комплекс апаратних та програмних засобів, що використовуються для створення, перетворення і виведення інформації у візуальному вигляді для комп'ютерних пристроїв відтворення. Також може бути, що комп'ютерну графіку бачать як сукупність великої кількості методів та прийомів перетворення комп'ютерних даних у графічні зображення.

Отже, саме з комп’ютерною графікою пов’язаний рендеринг зображень та метод трасування променів.

.

1. АНАЛІЗ ОСНОВНИХ МЕТОДІВ РЕНДЕРИНГУ ЗОБРАЖЕНЬ

1.1. Аналіз існуючих методів побудови зображення.

Для рендерингу зазвичай використовують 4 основні обчислювальні методи. Кожна техніка візуалізації має свій власний набір переваг і недоліків. Серед них існують:

- Scanline rendering and rasterization
- Ray casting
- Ray tracing
- Radiosity
- Z-buffer

Scanline rendering – це алгоритм визначення видимої поверхні в комп'ютерній графіці, працюючий на основі рядків за рядками замість полігонів та пікселів. Кожен багатокутник, котрий підлягається візуалізації, спочатку сортують за верхніми координатами, у яких вони створюються спочатку, а потім кожен рядок сканування зображення обчислюється з використанням перетину лінії сканування з полігонами на передній частині відсортованого списку. Далі цей список оновлюється, щоб відкинути невидимі полігони, оскільки активна лінія сканування просувається вниз. [1]

Переваги:

- Сортування вершин вздовж нормальної площини сканування зменшує кількість порівнянь між ребрами;
- Відсутність потреби переводу координат всіх вершин з основної пам'яті в робочу;
- Можливість інтегрування алгоритму з багатьма іншими графічними методами, такими як модель відображення Фонга або алгоритм Z-

буфера;

Недоліки:

- За допомогою даного методу немає можливості отримати реальні відображення та заломлення;
- Застарілість алгоритму;
- Поступається альтернативним методам у реалістичному рендерингу зображення;
- Складна реалізація, оскільки система працює з об’єктним кодом;

Ray casting – це алгоритм прокладання променів, у якому змодельована геометрія аналізує кожен рядок та кожен піксель з точки зору назовні об’єкта. У місці, де перетинається об’єкт, значення кольору в точці може бути оцінено за допомогою декількох методів. У найпростішому випадку значення кольору об’єкта в точці перетину стає значення цього пікселя. Також колір може бути визначений з текстурної карти. Більш складним методом є зміна значення кольору за допомогою коефіцієнта освітленості, але без розрахунку відношення до імітованого джерела світла. Інший метод робить розрахунок на кут падіння світлових променів від джерела світла, а також виходячи із зазначених інтенсивностей джерел світла, обчислює значення пікселя. [2]

Переваги:

- Легко реалізувати;
- Інтуїтивно створює алгоритм Line of Sight;

Недоліки:

- Повільний порівняно з іншими методами;
- При відливанні лише декількох променів, квадрати поблизу джерела будуть відвідуватися багато разів;
- Багато артефактів, навіть у звичайних ситуаціях;

Radiosity – метод радіосигналу синтезу зображень у середині 1980-х років. Система дивиться виключно на баланс світла або енергії в такому закритому середовищі, в якому вся енергія, що випромінюється або відбивається від даної поверхні, враховується шляхом відображення або поглинанням іншими поверхнями. За допомогою цього методу можна визначити величину поверхневого радіозв'язку та швидкість з якою енергія виходить з поверхні. Для обчислення радіозв'язку для кожної поверхні використовуються значення кількості взаємодій енергії між ними. Зазвичай, за допомогою відповідних маніпуляцій, радіосигнал може генерувати зображення на льоту на відстані 15-20 кадрів в секунду. Розрахунки радіозв'язку не залежать від точки зору але збільшують обчислення, що корисні для будь-яких точок зору. Якщо відбувається невелика перестановка об'єктів радіозв'язку в сцені, однакові дані радіосигналу можуть бути повторно використані для ряду кадрів, що робить радіозв'язок ефективним способом поліпшення відливання від променів, без серйозного впливу на загальний час візуалізації. Через це радіозв'язок є основним компонентом провідних методів візуалізації в реальному часі і використовується від початку до кінця для створення великої кількості відомих останніх анімаційних 3D-мультфільмів. [3]

Переваги:

- обчислює дифузні взаємозв'язки між поверхнями;
- забезпечує перегляд незалежних рішень для швидкого відображення довільних переглядів;
- відтворює відносно реалістичні зображення;

Недоліки:

- 3D сітка вимагає більше пам'яті, ніж оригінальні поверхні;
- алгоритм відбору проб поверхні є більш сприятливим до артефактів зображення, ніж трасування променів;

- не враховує дзеркальних відображень або ефектів прозорості

Z-buffer – алгоритм, який використовується для визначення видимої поверхні та є основою процесу візуалізації сканованої ліній. Головна ідея використання Z-buffer полягає в тому, щоб перевірити відстань від спостерігача кожної поверхні задля розробки найближчої поверхні кожного об’єкта. Якщо два об’єкти мають різні значення z-глибини вздовж однієї проектованої лінії, то вище значення знаходиться попереду, а позаду залишається ближча поверхня або об’єкт. Застосування цього підходу дозволяє нам відображати сцени за допомогою візуалізації сканової лінії. [4]

Переваги:

- простий у використанні;
- може бути легко реалізований в об’єкті або зображенні;
- може виконуватися швидко, навіть з багатьма полігонами;

Недоліки:

- займає багато пам’яті;
- неможливо створити прозорі поверхні без додаткового коду;

1.2. Обґрунтування теми дипломного проекту

Темою дипломної роботи було обрано створення системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя, що має такі переваги:

- Елегантність алгоритму;
- Гарна апроксимація відображень;
- Здатність працювати з великою кількістю явищ;
- Найреалістичніший і бажаний режим візуалізації на сьогоднішній день;
- Імітування руху світла в реальному світі;

- Точність надання прямого освітлення, тіней, дзеркального відображення та ефекту прозорості. Ефективна пам’ять;

У методі трасування променя, кожен піксель сцени, один або більше променів світла відстежуються від камери до найближчого 3D-об’єкста. Світловий промень проходить через задане число “відскоків”, що може включати відбиття або заломлення в залежності від матеріалу у 3D-сцені. Кожен колір пікселів обчислюється алгоритмічно на основі взаємодії світлового променя з об’єктами на його трасі. Ray tracing здатний більшого фотореалізму ніж інші методи рендерингу зображення, але його єдиним недоліком є складна обчислювальна спроможність.

1.3. Обґрунтування вибору середовища розробки

Для створення системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя за допомогою алгоритму ray tracing було обрано мову C++. C++ є досить портативною, а також об’єктно-орієнтованою мовою програмування і включає такі поняття, як класи, успадкування, поліморфізм, абстракція даних та інкапсуляція, які дозволяють повторно використовувати код і робить програми сприятливими до корегування.

Мова C++ надає користувачеві повний контроль над управлінням пам’ятю, що підвищує відповідальність користувача за керування пам’ятю. Забезпечує широкий спектр застосувань: від додатків графічного інтерфейсу до 3D-графіки та математичного моделювання, з чим ми і будемо працювати. Найбільшою перевагою мови C++ є її масштабність, тому програми, які зазвичай є ресурсо-місткими, зазвичай будуються з нею. Також C++ сумісна з багатою кількістю платформ, та підтримує C, тобто кожна діюча програма C є дійсною програмою C++.

Такий підхід використовують найкращі організації у сфері

комп’ютерної графіки та ігор, і процес рендерингу зображення прогресує з кожним днем, наприклад, графіку у сучасних комп’ютерних іграх вже можливо сплутати з реальним життям.

1.4. Висновки

В цьому розділі були розібрані основні існуючі методи побудови зображення та їх переваги та недоліки. Був обгрунтований вибір теми дипломного проекту, та пояснено домінування обраного методу рендерингу зображення відносно альтернативних методів. Також детально розібрали причини вибору конкретного середовища розробки.

2. ТЕХНІКА ВІЗУАЛІЗАЦІЇ ВІДСТЕЖЕННЯ ПРОМЕНІВ

2.1. Огляд техніки візуалізації відстеження променів

Трасування променів називають метод обчислення видимості між точками. Транспортні алгоритми призначені для імітації способу поширення світла через простір при взаємодії з об’єктами. Їх використовують для обчислення кольору точки сцени. Трасування променів не є легким транспортним алгоритмом, це лише техніка обчислення видимості між точками. [4]

Растове зображення зроблено з пікселів. Один із способів відтворення 3D-сцени полягає в тому, щоб якось рухати це растове зображення вздовж площини зображення віртуальної камери, і знімати промені (Рис 2.1).

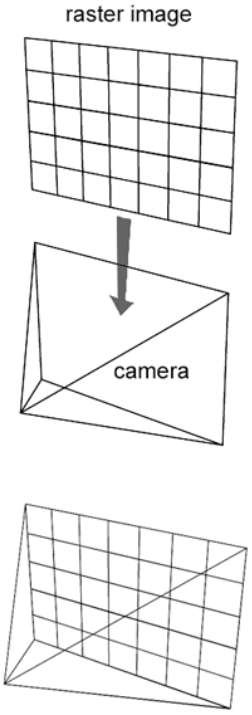

Рис. 2.1 – рух растового зображення перед площиною зображення камери [5]

За допомогою закидання променів, що походять від ока (положення камери), та які проходять через центр кожного пікселя ми знаходимо об’єкт

зі сцени на якому ці промені перетинаються.

Якщо піксель щось і «бачить», то він бачить саме об’єкт, що знаходиться прямо по напрямку, вказаного променем. Напрямок променя можна побудувати, якщо простежити лінію від походження камери до центру пікселя (Рис 2.2).

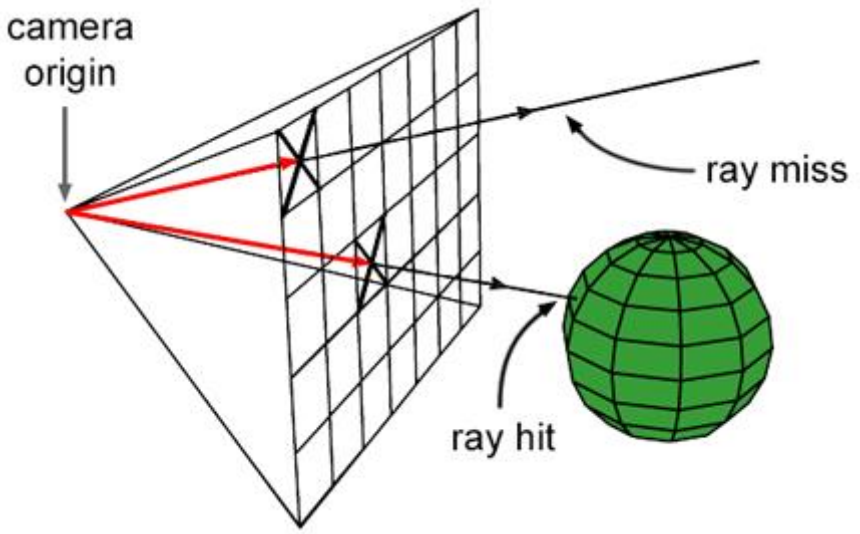

Рис. 2.2 – промінь може потрапити в геометрію сцени або пропустити її [5]

Тепер, коли ми знаємо, що бачить піксель, треба лише повторити цей процес для кожного пікселя зображення. Налаштовуючи колір пікселя відповідно до кольору об’єкта, в котрому кожен промінь проходить через центр кожного пікселя, ми можемо сформувати зображення сцени. Цей метод вимагає циклічного перегляду всіх пікселів у зображенні і і перетворення променя на сцену для кожного з цих пікселів. На другому етапі, етапі перетину, потрібно виконати цикл над усіма об’єктами сцени, щоб перевірити, чи перетинає промінь будь-яких з цих об’єктів. Нижче приведена реалізація цієї техніки в псевдокоді (Псевдокод 2.1).

```
Vec3f *framebuffer = new Vec3f[imageWidth * imageHeight];
for (int j = 0; j < imageHeight; ++j) {
```

```
for (int i = 0; i < imageWidth; ++i) {
for (int k = 0; k < numObjectsInScene; ++k) {
Ray ray = buildCameraRay(i, j);
if (intersect(ray, objects[k]) {
// тут зробити комплексне затінення, але для основного кольору
framebuffer[j * imageWidth + i] = objects[k].color;
}
else {
// або нічого не робимо і залишаємо його чорним
framebuffer[j * imageWidth + i] = backgroundColor;
}
}
}
}
```

Псевдокод 2.1 - цикл над усіма пікселями

Слід зауважити, що деякі промені можуть взагалі не перетинати будь-яку геометрію. Наприклад, як показано на малюнку 2.2, один з променів не перетинає сферу. У цьому випадку, як правило, треба залишити чорний колір пікселя, або встановити його на довільний інший колір. В наведеному вище коді ми ми задаємо колір пікселя кольором об’єкта в точці перетину.

Об’єкти в реальному світі виглядають дуже складно. Їх яскравість змінюється залежно від кількості світла, яке вони отримують, деякі з них блискучі, інші матові тощо. Мета фото-реалістичної візуалізації полягає не лише у точному зображенні геометрії, але й в імітуванні зовнішнього вигляду об’єктів переконливо (Рис 2.3).

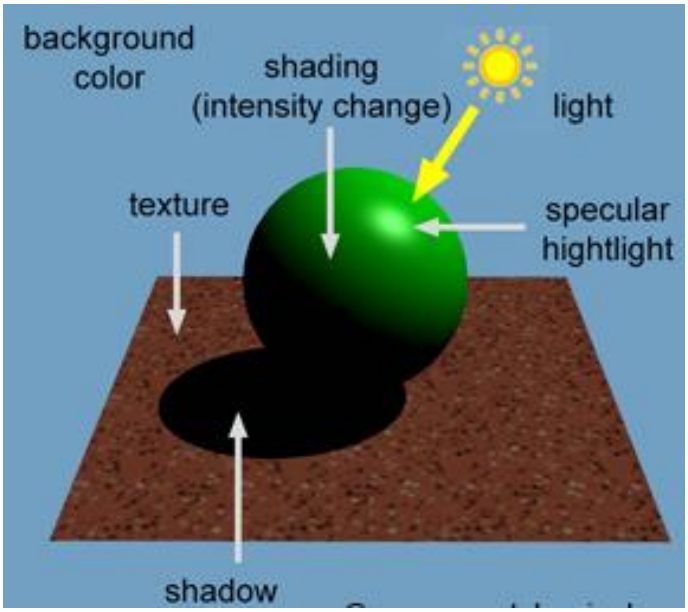

Рис. 2.3 – ілюстрація, що показує, як важко відтворити реальність [6]

Цей процес передбачає щось більш складне, ніж просто повернення постійного кольору. У комп'ютерній графіці, завдання визначення фактичного кольору об'єкта в будь-якій заданій точці на його поверхні, знаючи про ймовірні зовнішні (кількість світла, що отримує об'єкт) та внутрішні параметри (матерія об'єкта, блискучі кольори), називається затіненням.

Трасування променів вважається орієнтованим на зображення. Зовнішні петлі повторюють всі пікселі зображення і внутрішній цикл перебирає об'єкти сцени. Алгоритм растеризації є відносно орієнтованим на об'єкт. Він вимагає циклічного перегляду всіх геометричних примітивів сцени та проектування цих примітивів на екран. Приведемо приклад швидкої реалізації циклів растеризації в псевдокоді (Псевдокод 2.2).

```
// цикл над усіма пікселями
Vec3f *framebuffer = new Vec3f[imageWidth * imageHeight];
for (int k = 0; k < numObjectsInScene; ++k) {
```

```
// проектування об’єкту на сцену, використовуючи перспективну проекцію.
...
for (int j = 0; j < imageHeight; ++j) {
for (int i = 0; i < imageWidth; ++i) {
if (pixelCoversGeometry(i, j, objects[k]) {
framebuffer[j * imageWidth + i] = objects[k].color;
}
}
}
}
```

Псевдокод 2.2 - швидка реалізації циклів растеризації

Трасування променів слід використовувати для двох етапів візуалізації: видимість та затінення. Це пов’язано з тим, що растеризація підходить тільки для видимості, і краща ніж трасування променів лише у швидкодії.

Зазвичай рендеринг займається обчисленням видимості між точкою в просторі і першою видимою поверхнею в заданому напрямку, або видимістю між двома точками. Перша задача призначена для вирішення проблеми видимості, друга - для вирішення таких проблем, як затінення. Растеризація дуже добре підходить для пошуку першої видимої поверхні, але неефективна для вирішення задачі видимості між двома точками. Трасування променів дозволяє ефективно обробляти обидва випадки. Пошук першої видимої поверхні корисний для вирішення проблеми видимості. Таким чином, і трасування променів, і растерізація підходять для вирішення цієї задачі. Затінення вимагає вирішення проблеми видимості між поверхнями, яке використовується для обчислення тіней,

коли використовуються світлові області, і глобальні ефекти освітлення, такі як відображення, заломлення, непрямі відображення та непрямі дифузії. Таким чином, для цієї конкретної частини процесу візуалізації трасування променів є більш ефективним, ніж растеризація. Але майте на увазі, що будь-яка техніка, яка обчислює видимість між точками, може бути використана для затінення та вирішення проблеми видимості.

2.2. Етапи обчислення фотореалістичних 3D-зображень за допомогою трасування променів.

Використання трасування променів для обчислення фотореалістичних зображень може бути поділено на 3 етапи:

- Відливання променів на сцену;
- Тестування перехресть променевої геометрії;
- Затінення;

Перше, що потрібно зробити, щоб створити зображення за допомогою трасуванням променів, це подати промінь для кожного пікселя зображення. Ці промені називаються камерами або первинними променями. Вторинні використовуються для пошуку точок сцени, що знаходяться в тіні, або для обчислення ефектів затінення, таких як відображення або заломлення. Коли первинний промінь відкидається в сцену, наступний крок полягає в тому, щоб знайти його перетин з будь-яким об’єктом у сцені.

Тестування променя на перетин з будь-яким об’єктом у сцені вимагає циклічного перегляду всіх об’єктів сцени і перевірки поточного об’єкта на промінь (Псевдокод 2.3).

```
Ray  ray = buildCameraRay(i,  j);
for  (int  k  =  0; k  <  numObjects; ++k) {
if  (intersect(ray,  objects[k]) {
// this  ray  intersects  objects[k]
```

```
}
}
```

Псевдокод 2.3 – циклічний перегляд всіх об’єктів сцени для перевірки поточного об’єкта на промінь

Геометрія в 3D може бути визначена різними способами. Прості форми, такі сфера, диски, площини, можуть бути визначені математично або параметрично. Форми більш складних об’єктів можуть бути описані лише за допомогою полігонових сіток, поверхонь підрозділів поверхонь NURBS. Основна проблема з другою категорією об’єктів полягає в тому, що для кожної підтримуваної поверхні необхідно реалізувати методи перетину променевої геометрії. Наприклад, поверхні NURBS можна промалювати прямо, хоча рішення для цього дуже відрізняється від того, що використовується для тестування перетину між променем і багатокутною сіткою. Таким чином, для того щоб підтримувати всі типи геометрії, треба написати одну з перехресних процедур променевої геометрії для кожного підтримуваного типу, але це потенційно збільшує складність коду програми. Альтернативне рішення полягає в перетворенні кожного типу геометрії в одне і те ж внутрішнє уявлення, яке майже завжди буде триангульованою сіткою багатокутника (Рис 2.4).

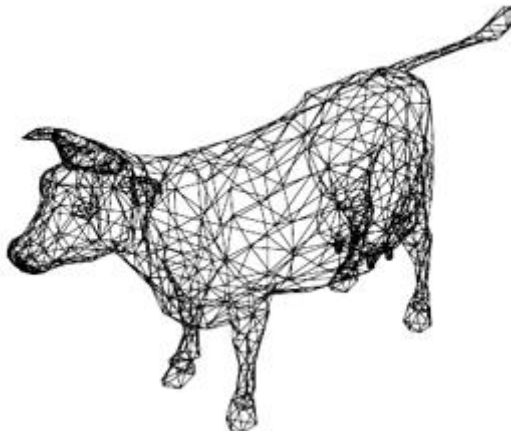

Рис. 2.4 – триангульовані сіткі зручніші для трасування променів, ніж

інші типи геометрії [8]

Трапляється, що перетворення майже будь-якого типу типу поверхні на сітку багатокутника дуже просте. Звідси перетворення полігональної сіткі в триангульовану також досить просте. Крім цього, трикутники є вигідними з той точки зору, що вони можуть бути використані як базовий геометричний примітив для трасування променів та растеризаації. Обидва алгоритми люблять трикутники через те, що вони мають цікаві геометричні властивості, яких не мають інші типи.

Таким чином, полігональні сітки або інші типи поверхонь перетворюються на трикутники. Це можна зробити або перед завантаженням геометрії в пам’ять програми, або під час візуалізації. Тепер не тільки кожен промінь камери повинен бути протестований на кожному об’єкті сцени, але і на кожному окремому трикутнику, який складається з кожного багатокутника в сцені. Цикл буде виглядати наступним чином (Псевдокод 2.4).

```
for (int j = 0; j < imageHeight; ++j) {
for (int i = 0; i < imageWidth; ++i) {
for (int k = 0; k < numObjects; ++k) {
for (int n = 0; n < objects[k]. numTriangles; ++n) {
Ray ray = buildCameraRay(i, j);
if (intersect(ray, objects[k] triangles[n])) {
framebuffer [j * imageWidth + i] = shade(ray, objects[k], n);
}
}
}
}
}
```

Псевдокод 2.4 – цикл для тестування променів на кожному об’єкті сцени та на кожному окремому трикутнику

Це означає, що час, необхідний для відтворення сцени трасування променів, прямо пропорційний кількості трикутників, які містить сцена. Всі трикутники повинні бути збережені в пам’яті, і кожен з них повинен бути перевірений на додавання променя в сцену. Висока обчислювальна вартість трасування променів є головним недоліком алгоритму.

2.3.Опис функції трасування

У трасуванні променів, внутрішні петлі (цикли над усіма об’єктами і всі трикутники, що містяться в кожному об’єкті) зазвичай переміщуються до функції, які зазвичай називається trace(). Функція повертає true у випадку, коли об’єкт або трикутник перетнулися, та в іншому випадку – false (Псевдокод 2.5).

```
bool trace(
const Vec3f& rayOrigin, const Vec3f &rayDirection,
const Object* objects, const uint32_t &numObjects,
uint32_t &objectIndex,
uint32_t & triangleIndex
)
{
bool intersect = false;
for (uint32_t k = 0; k < numObjects; ++k) {
for (uint32_t n = 0; n < objects[k]. numTriangles; ++n)) {
if (rayTriangle Intersect(rayOrigin, rayDirection, objects[k]. triangles[n]) {
intersect |= true;
objectIndex = k;
```

```
triangleIndex = n;
}
}
}

return intersect;
}

int main(...)
{
...
uint32 t objectIndeix, triangleIndex;
for (int j = 0; j < imageHeight; ++j) {
for (int i = 0; i < imageWidth; ++i) {
if (trace(rayOrigin, rayDirection, objects, numObjects, objectIndex,
triangleIndex)) {
framebuffer[j * imageWidth + i] = shade (rayOrigin, rayDirection, objects
[objectIndex], triangleIndex);
}
}
}
...
return 0;
}
```

Псевдокод 2.5 – функція трасування

На малюнку (Рис 2.5) можна побачити, як промінь перетинає кілька трикутників. У випадку растеризації, ми вирішуємо цю проблему, використовуючи буфер глибини, а для трасування променів достатньо

просто створити змінну в функції trace(), яка повинна відстежувати найближчу відстань між джерелом променів і точкою перетину.

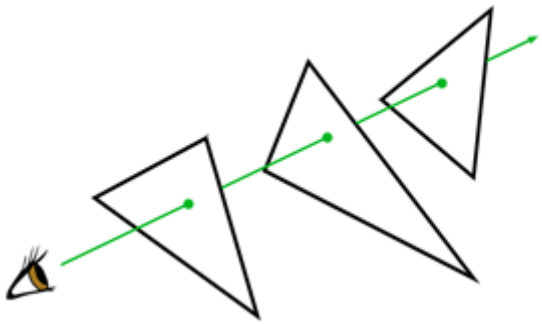

Рис. 2.5 – промінь, який перетинає кілька об’єктів [9]

Коли знаходиться перетин, відстань L до трикутника звіряється зі змінною $L_{найближча}$, яка спочатку встановлена як нескінченність. Якщо L менше за $L_{найближча}$, то у значення $L_{найближча}$ встановлюється значення L, і коли всі трикутники всіх об’єктів перевірені, $L_{найближча}$ буде містити відстань до найближчого трикутника. Відстань L, що визначає проміжок від точки вздовж променя до походження промення буде описана наступним чином в псевдокоді (Псевдокод 2.6).

```
Vec3f P = rayOrigin + t * rayDirection;
```

```
bool trace(
const Vec3f& rayOrigin, const Vec3f &rayDirection,
const Object* objects, const uint32_t &numObjects,
uint32_t &objectIndex,
uint32_t &triangleIndex,
float &tNearest
)
{
tNearest = INFINITY; // initialize to infinity
```

```
bool intersect = false;
for (uint32_t k = 0; k < numObjects; ++k) {
for (uint32_t n = 0; n < objects[k].numTriangles; ++n) {
float t;
if (rayTriangle Intersect(rayOrigin, rayDirection, objects[k].triangles[n], t)
&& t < tNearest) {
objectIndex = k;
triangleIndex = n;
tNearest = t; // оновлення точки перетину до найближчої з ними методом
перевірки попередньої
intersect |= true;
}
}
}

return intersect;
}
```

Псевдокод 2.6 – визначення відстані від точки вздовж променя до походження променя

Відстань L, що визначає проміжок від точки вздовж променя до походження променя буде описана наступним чином в псевдокоді.

2.4. Аналіз етапу затінення

Хоча трасування променів має високу обчислювальну вартість, в порівнянні з іншими підходами, воно має багато переваг, що роблять його набагато привабливішим, це особливо стосується затінення.

Після знаходження об’єкту, який перетинає промінь, треба з’ясувати, який колір об’єкта знаходиться в точці перетину. Колір предметів або їх інтенсивність зазвичай відрізняються по всій поверхні. Це зазвичай викликано зміною освітлення, а також тим, що текстура об’єктів змінюється по всій поверхні. Колір об’єкта в будь-якій точці його поверхні є істинним результатом того, як об’єкт повертається або відображає світло, що чіпляє його поверхню відносно спостерігача. Колір в точці перетину залежить від:

- Кількості світла, що впливає на поверхню об’єкта;
- Напрямок цього світла;
- Позиція спостерігача;

У природі, процес затінення можна охарактеризувати як «світлові промені», які випускаються з джерела світла та падають на поверхню об’єктів, після чого відбиваються назад на сцену. Деякі з цих відбитих променів будуть вражати більше поверхонь і будуть відображені знову по черзі. Інші промені можуть потрапити в очі спостерігача або камеру. Трасування променів простежує шлях світлових променів назад від ока до поверхні, а потім від поверхні до джерела світла, що визначає процес зворотнього відстеження (Рис 2.6).

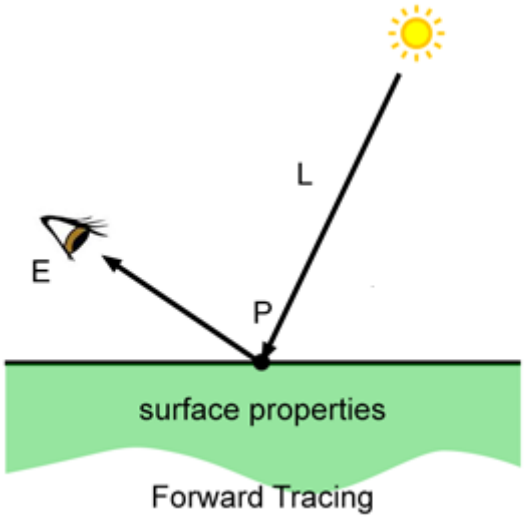

Рис. 2.6 – світло, що відбивається від поверхні P на спостерігача [10]

У очевидному підході до трасування променів, промені світла, що виходять з джерела, простежуються по їх шляхах, поки вони не вражають глядача, але оскільки лише деякі з них досягнуть глядача, такий підхід є марнотратним. У другому підході, запропонованому Аппелем, промені простежуються в протилежному напрямку – від глядача до об'єктів у сцені. [5].

Функція легкого транспортного алгоритму полягає в моделюванні способу поширення світла у вигляді енергії через сцену. Матеріал, який відбивається світлом у навколишнє середовище, визначається математичною моделлю, або моделлю освітлення в комп'ютерній графіці. Модель Фонга є прикладом моделі освітлення. Моделі освітлення зазвичай реалізуються в Шейдерах. Шейдери, як правило, реалізуються як деяка функція, яка приймає вхідний напрямок падаючого світла W_i, інтенсивність, напрямок, властивості матеріалу. Та кількість падаючого світла, що відображається в напрямку W_o (Рис 2.7).

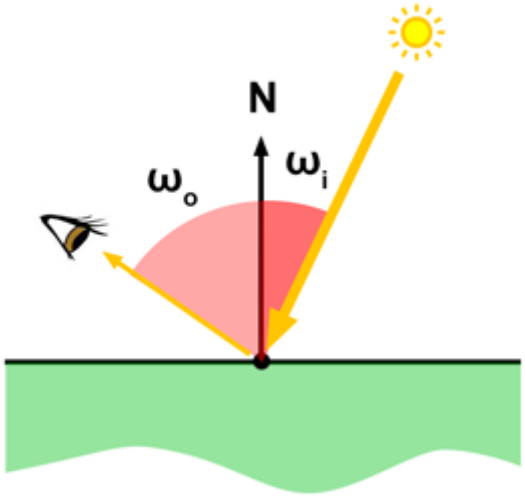

Рис. 2.7 – функція моделі освітлення полягає у визначенні кількості світла, що відображається у напрямку W_o на основі властивостей поверхні, таких як колір, інтенсивність світла і напрямок падаючого світла [11]

Затінення включає в себе моделювання способу поширення світлової енергії через сцену. Одним з найбільш природних способів розгляду цієї проблеми є світлові промені. Наприклад, треба слідувати шляху світлового променя, який випромінюється джерелом світла, а потім відбити його дифузним об’єктом у напрямку до дзеркала, що відіб’є промінь на наше око (Рис 2.8).

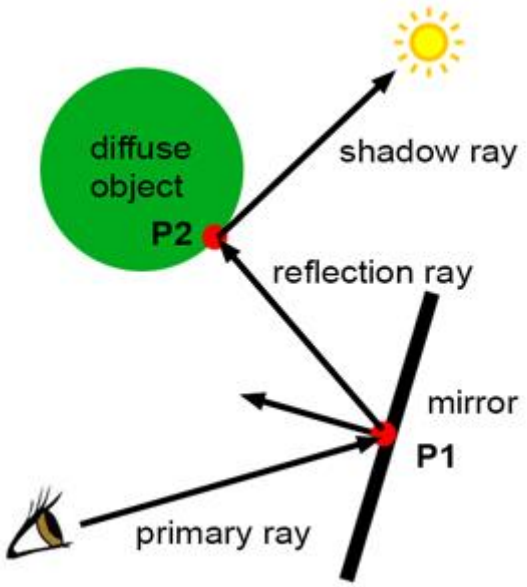

Рис. 2.8 – відображення первинних, вторинних та тіньових променів [11]

Оскільки в CG вважається, що краще використовувати зворотнє відстеження, можна легко обчислити напрямок відбиття променя, використовуючи закон відображення. Всі промені, які кидаються після первинного, називаються вторинними променями. Наприклад, промені відбиття та тіньові промені є перш за все вторинними. Головною метою цих променів є знаходження, чи знаходиться точка на дифузному об’єкті в тіні світла. Кожен з вторинних променів передбачає обчислення видимості між двома точками.

2.5.Висновки

Отже, в цьому розділі було детально розглянуто техніку візуалізації відстеження променів та методи обчислення видимості між точками. Було

продемонстровані реалізації методів обчислення видимості у псевдокоді. Також були розібрані етапи трасування променів для обчислення фотореалістичних зображень тривимірних об’єктів, такі як відливання променів на сцену, тестування для перехресть променевої геометрії та затінення. Після цього детально пояснили функціонування функції трасування.

3. СТРУКТУРА МЕТОДУ ПОБУДОВИ ЗОБРАЖЕННЯ ЗА ДОПОМОГОЮ ЗВОРОТНЬОГО ВІДСТЕЖЕННЯ ПРОМЕНЯ

3.1. Аналіз алгоритму пропускання променів

Людині знадобилося багато часу для дослідження та розуміння природи світла. Греки розробили теорію зору, в якій об’єкти бачилися променями світла, що виходять з очей. Арабський учений, Ібн-Аль-Хайтам, першим пояснив, що ми бачимо об’єкти через сонячні промені світла, як потоки крихітних частинок, що подорожують по прямих лініях та відбиваються від предметів в наших очах, формуючи зображення (Рис 3.1).

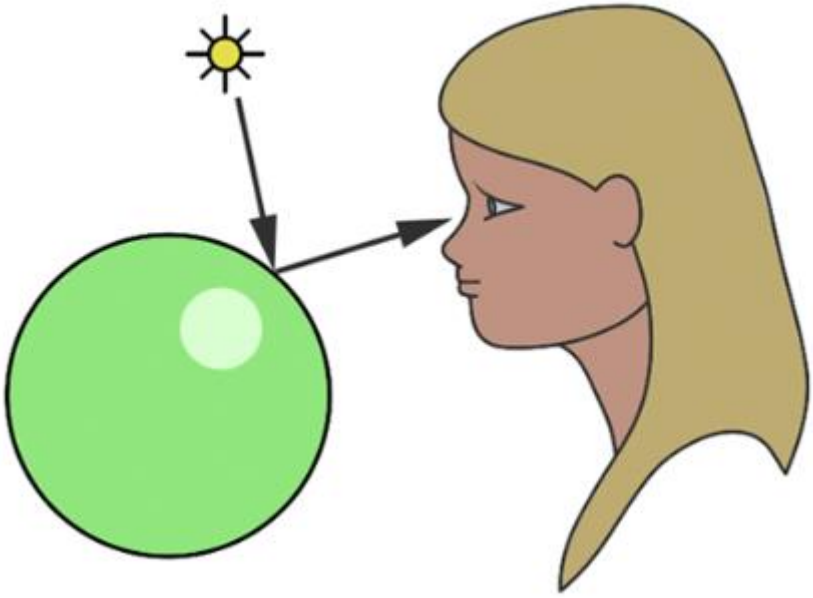

Рис. 3.1 – Модель Аль-Хайтама [12]

Явища, описані Ібн-Аль-Хайтамом, пояснюють, чому ми бачимо об’єкти. На його спостереженнях можна зробити два цікавих зауваження: по-перше, без світла ми нічого не бачимо, по-друге, без предметів у нашому середовищі ми не зможемо побачити світло. Якщо навколо нас

немає матерії, ми не можемо побачити нічого, крім темряви, навіть якщо фотони потенційно рухаються через цей простір.[6]

У порівнянні з загальною кількістю променів, що відбиваються об’єктом, лише деякі з них можуть досягти поверхні нашого ока. Наприклад, створимо джерело світла, яке випускає тільки один фотон за один раз. Він буде випромінюватися джерелом світла та проходити по прямій лінії, поки не потрапить на поверхню нашого об’єкта. Ігноруючи поглинання фотонів, можна припустити, що фотон відбивається у випадковому напрямку. Якщо фотони потрапляють на поверхню нашого ока, то ми бачимо точку, де відбивався фотон (Рис 3.2).

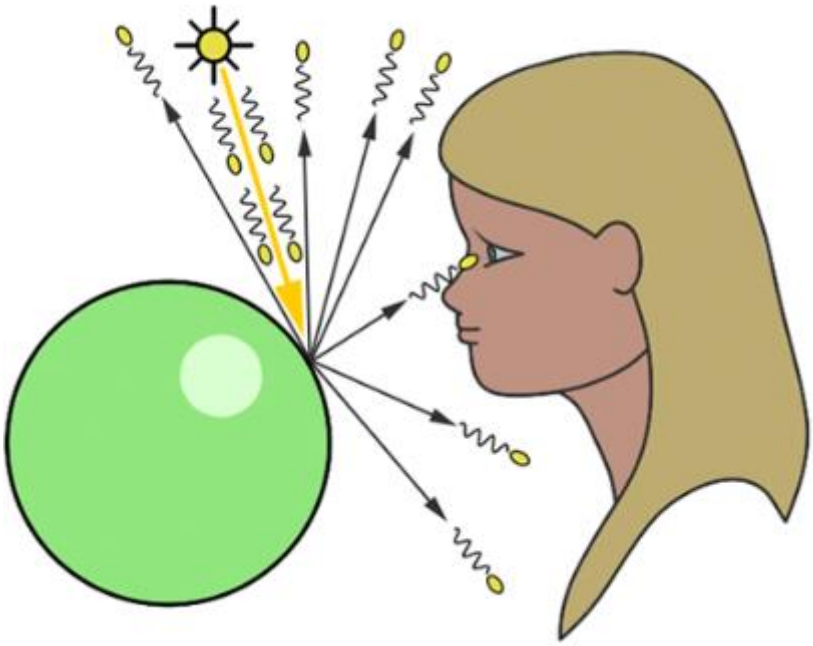

Рис. 3.2 – незліченні фотони, що випускаються джерелом світла, потрапляють в зелену сферу, але тільки один досягає поверхні ока [12]

З точки зору комп’ютерної графіки, ми замінюємо наші очі площиною зображення, складеною з пікселів. У цьому випадку фотони, що випромінюються, потравляють в один з багатьох пікселів на площині зображення, збільшуючи яскравість в цій точці до значення більше нуля.

Цей процес повторюється кілька разів, поки всі пікселі не налаштовані, створюючи комп'ютерно-генерований образ. Цей метод називається прямим трасуванням променів, оскільки ми слідуємо по шляху фотона вперед від джерела світла до спостерігача. Трасування прямого променя робить технічно можливим моделювати шлях світла в природі на комп'ютері. Однак цей метод не є ефективним або практичним, і саме зворотнє трасування радить один з перших дослідників комп'ютерної графіки Тернер Уитліс (Рис 3.3).

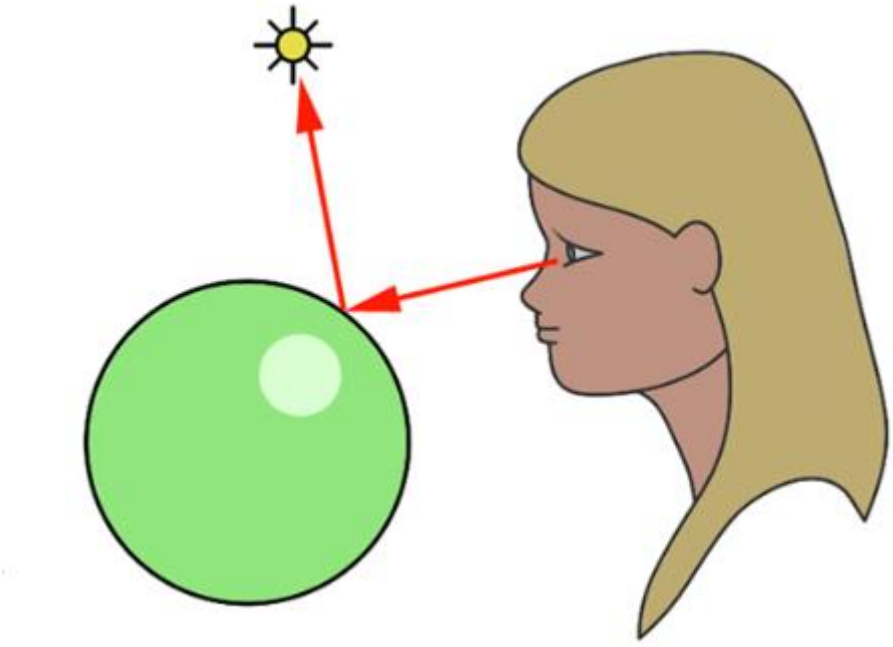

Рис. 3.3 – зворотнє трасування променів. Простежуємо промінь від ока до точки на сфері, а потім промінь від цієї сфери до джерела живлення [12]

Оскільки наше моделювання не може бути таким швидким і досконалим, як природа, ми повинні скомпрометувати і простежити промінь від ока до сцени. Якщо промінь потрапляє на об'єкт, можна виявити, скільки світла він отримує, кидаючи інший промінь від точки попадання до світла сцени. Іноді цей промінь закривається іншим об'єктом

зі сцени, що означає, що наша первісна точка попадання знаходиться в тіні та не отримує ніякого освітлення від світла.

3.2.Реалізація алгоритму Ray Tracing

Перш за все, поширення світла в природі – це незлічена кількість променів, які випромінюють джерела світла, що обертаються, поки вони не потрапляють на поверхню нашого ока. Трасування променів елегантне в тому, що воно засновано безпосередньо на тому, що насправді відбувається навколо нас. Крім того, що воно йде по шляху світла в зворотному порядку, це ще й ідеальний симулятор природи. [7]

Алгоритм трасування променів приймає зображення з пікселів. Для кожного пікселя зображення він знімає первинний промінь у сцени. Напрямок цього променя досягається шляхом відстеження лінії від ока до центру цього пікселя. Після встановлення напрямку первинного променя, перевіряється кожен об’єкт сцени, щоб побачити, чи перетинається він з будь-яким з них. У деяких випадках первинний промінь перетинає більше одного об’єкта. У такому випадку, вибирається об’єкт, точка перетину якого найближче до ока. Потім знімається тіньовий промінь від точки перетину до світла (Рис 3.4).

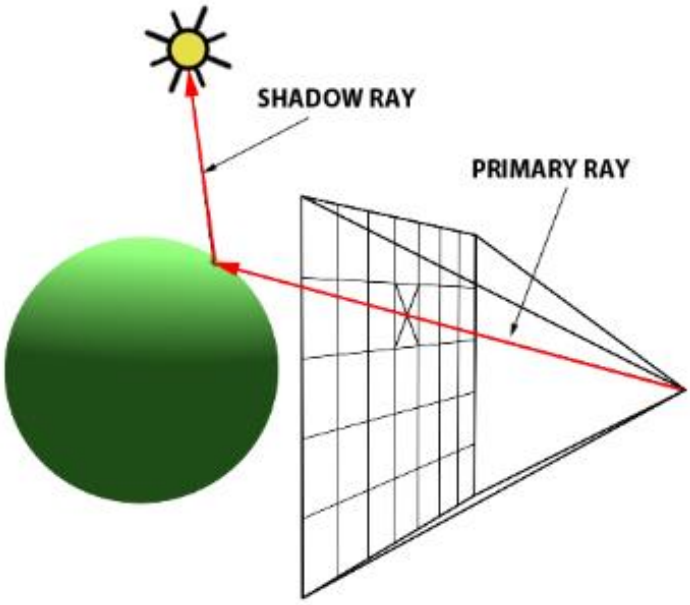

Рис. 3.4 – знімається первинний промінь через центр пікселя для перетину можливого перетину об’єкта, та для точки перетину кидається тіньовий промінь, щоб дізнатися, освітлена ця точка або в тіні [10]

Якщо цей промінь не перетинає об’єкт на шляху до світла, то точка влучення висвітлюється. Якщо він перетинається з іншим об’єктом, то цей об’єкт на ньому відкидає тінь (Рис 3.5).

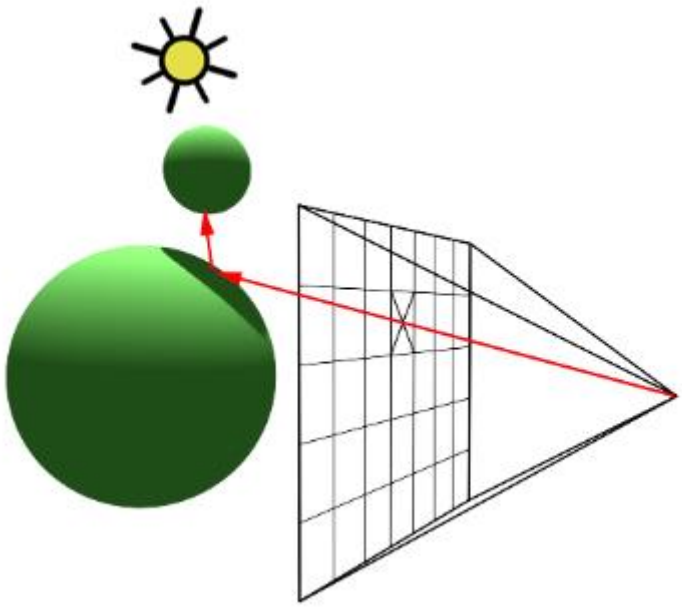

Рис. 3.5 – маленька сфера кидає тінь на велику сферу. Тіньовий промінь перетинає сферу, перш ніж він потрапить до світла [10]

Якщо повторити цю операцію для кожного пікселя, то можна отримати двовимірне подання нашої тривимірної сцени (Рис 3.6).

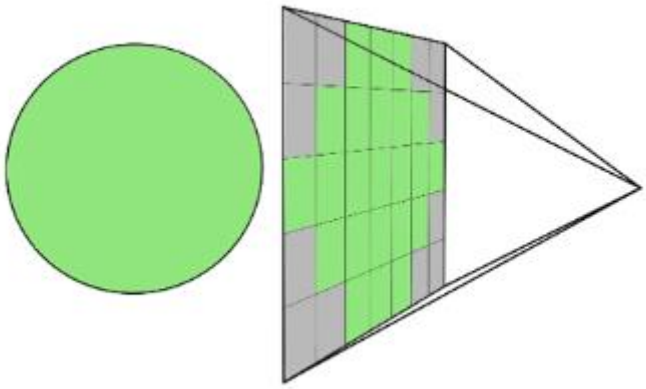

Рис. 3.6 – для візуалізації кадру, знімається первинний промінь для кожного пікселя буфера кадру [10]

Нижче приведена реалізація цього алгоритму в псевдокоді (Псевдокод 3.1).

```
    for (int j = 0; j < imageHeight; ++j) {
for (int i = 0; i < imageWidth; ++i) {
// обчислюємо первинний напрямок променя
Ray primRay;
computePrimRay(i, j, &primRay);
// знімаємо первинний промінь у сцені та шукаємо перехрестя
Point pHit;
Normal nHit;
float minDist = INFINITY;
Object object = NULL;
for (int k = 0; k < objects.size(); ++k) {
if (Intersect(objects[k], primRay, &pHit, &nHit)) {
float distance = Distance(eyePosition, pHit);
```

```
if (distance < minDistance) {
object = objects[k];
minDistance = distance; // оновлення мінімальної дистанції
}
}
}
if (object != NULL) {
// обчислюємо освітлення
Ray shadowRay;
shadowRay.direction = lightPosition - pHit;
bool isShadow = false;
for (int k = 0; k < objects.size(); ++k) {
if (Intersect(objects[k], shadowRay)) {
isInShadow = true;
break;
}
}
}
if (!isInShadow)
pixels[i][j] = object->color * light.brightness;
else
pixels[i][j] = 0;
}
}
```

Псевдокод 3.1 – реалізація алгоритму Ray Tracing

Одна з найважливіших переваг трасування променів, полягає в тому, що кодування цього алгоритму потребує малих зусиль для реалізації, на відміну від інших алгоритмів, таких як візуалізатор сканування.

Ця методика була вперше описана Артуром Аппелем в 1969 році. Статтею під назвою «Деякі методики відтінку твердих тіл» [9]. Цей алгоритм не замінив всі інші алгоритми візуалізації тому, що в той час він був дуже повільним, та в деякій мірі випереджав технології. Надзвичайно багато часу, щоб знайти перетин між променями і геометрією. Протягом десятиліть швидкість алгоритму була основним недоліком трасування променів. Однак, оскільки комп'ютери стають швидшими, це все менше і менше. На сьогоднішній день, за допомогою швидких комп'ютерів, можна обчислити кадр, який займав одну годину протягом декількох хвилин або швидше. [10].

3.3.Додавання відображення та заломлення

Іншою перевагою трасування променів є те, що, розширюючи ідею розповсюдження променів, можна легко моделювати ефекти, такі як відбиття і заломлення, які зручні для імітації скляних матеріалів або дзеркальних поверхонь.

У документі 1979 року під назвою «Покращена модель освітлення для зафарбованого дісплея», Тернер Уіттіс був першим, хто описав, як розширити алгоритм трасування променів Аппеля більш просунутого рендеринга. Ідея Вітіта розширила модель зйомки променів Аппеля для включення обчислень як для відображення, так і для заломлення [11].

У оптиці добре відомі явища відображення і заломлення. Найбільш доцільним прикладом для розглядання буде скляна кулька, що має як

рефракційні, так і відбивні властивості. Доки ми знаємо напрямок променя, що перетинає кулю, легко обчислити, що з ним відбувається. Напрями відбиття і заломлення засновані на нормі в точці перетину і напрямку вхідного променя (первинного). Для обчислення напрямку рефракції необхідно також вказати індекс заломлення матеріалу. Рефракцію можна візуалізувати, коли промень зігнутий. Коли фотон потрапляє на об'єкт іншого середовища, його напрямок змінюється.

Отже, знаючи, що скляна куля є одночасно рефлекторною і заломлюючою, треба обраховувати дві точки для певної точки на поверхні. Змішування значень залежить від кута між первинним променем та нормальним об'єктом і індексом заломлення. На щастя, в науці існує рівняння, яке точно розраховує, як кожне значення провинно бути змішане. Це рівняння відоме як рівняння Френеля.

Алгоритм Whitted працює таким чином, що треба знімати первинний промінь з ока і найближчого перетину з об'єктами сцени. Якщо промінь вражає об'єкт, який не є дифузним або прозорим, потрібно зробити додаткову обчислювальну роботу. Щоб обчислити отриманий колір у цій точці, наприклад, скляну кулю, потрібно обчислити колір відбиття і колір заломлення, після чого змішати їх разом. Все це робиться в три кроки. Обчислення кольору відбиття, обчислення кольору заломлення, а потім застосування рівняння Френеля. [14]

Спочатку обчислюється напрямок відображення. Для цього нам потрібні 2 значення: норма в точці перетину і напрямок первинного променя. Як тільки отримуємо напрямок відбиття, знімається новий промінь у цьому напрямку. Наприклад, промінь відбиття потрапляє на червону кулю (Рис 3.7).

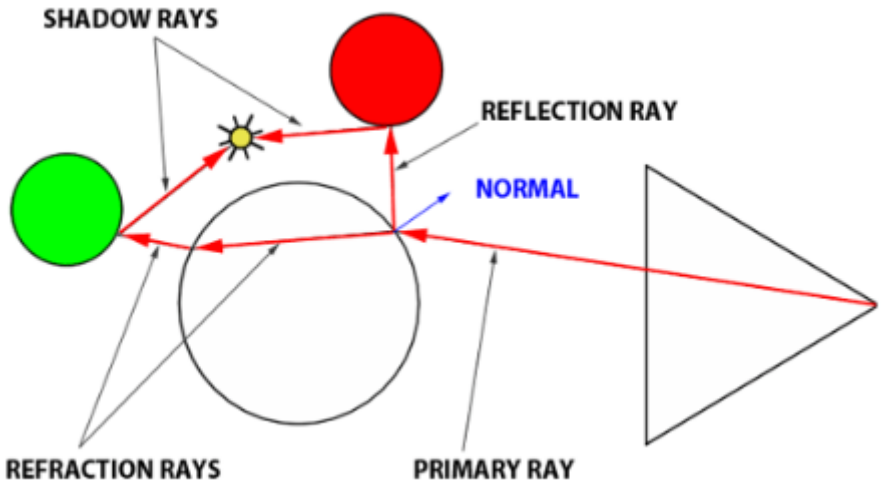

Рис. 3.7 – використання оптичних законів для обчислення відбиття та рефракційних променів [14]

Використовуючи алгоритм Аппеля, можна виявити, скільки світла досягає цієї точки на червоній сфері, стріляючи променем тіні до світла. Цей колір отримує чорний колір, якщо він затінений, який потім помножується на інтенсивність світла і повертається до поверхні скляної кулі.

Тепер можна зробити ту саму операцію для рефракції. Промінь, який проходить через скляну кулю, називається променем пропускання. Щоб обчислити напрямок передачі, потрібна норма в точці попадання, первинний напрямок променя та показник заломлення матеріалу (наприклад, для матеріалу скла це може бути близько 1,5). Коли новий напрямок обчислено, рефракційний промінь продовжує рухатися до іншого боку скляної кулі. І промінь знову переломлюється, тому що він змінює середовище. Як можна побачити на малюнку 3.8, напрямок променя змінюється, коли промінь входить і виходить зі скла (Рис 3.8).

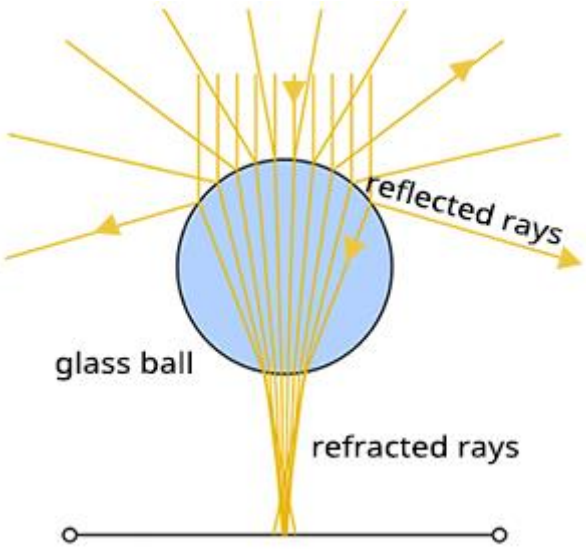

Рис. 3.8 – ілюстрація проходження рефракційних променів скрізь скляну кулю [14]

Рефракція відбувається кожного разу, коли змінюється середовище. Два середовища, з яких виходить промінь і потрапляє, мають два різних показника заломлення. Показник заломлення повітря дуже близький до 1, а для повітря близько 1,5. Рефракція має злегка згинати промінь для поліпшення ефекту. Цей процес робиться для того, щоб об’єкти здавалися зміщеними при перегляді

Отже, для обчислення рівняння Френеля, нам потрібен показник заломлення скляної кульки та кут між первинним променем і нормальним в точці попадання. Рівняння Френеля повертає два значення змішування. Приведемо приклад псевдокоду, для підсилення роботи (Псевдокод 3.2).

```
// розрахунок кольору відображення
color reflectionCol = computeReflectionColor();
// розрахунок кольору заломлення
color refractionCol = computeRefractionColor();
float Kr; // reflection mix value
float Kt; // refraction mix value
```

```
fresnel (refractiveIndex, normalHit, primaryRayDirection, &Kr, &Kt);
// змішуємо обидва
color glassBallColorAtHit = Kr * reflectionColor+ (1- Kr) *
refractionColor;
```

Псевдокод 3.2 – рівняння Френеля

Великою перевагою в цьому алгоритму є те, що він рекурсивний. У прикладі ми розглядали, що промінь відбиття потрапляє в червону, непрозору сферу, а рефракція проникає в зелену, непрозору і дифузну сферу. Тепер треба уявити, що червона і зелена сфера – це також скляні кулі. Щоб знайти колір, що повертається відбиттям і рефракційними променями, треба слідувати тим же процесом з червоними і зеленими сферами, який використовувався для оригінальної скляної кулі. Це є невеликим недоліком алгоритму трасування променів. Кожного разу, коли промінь відбивається або заломлюється, його глибина збільшується. Для цього треба зупиняти процес рекурсії, коли глибина променя досягає максимальної глибини рекурсії.

3.4. Висновок

У комп’ютерній графіці концепція зйомки променів або світла, або від очей називається трасуванням траси. Термін трасування променів також може бути використаний, але концепція трасування траси дозволяє припустити, що цей спосіб створення зображень, що генеруються комп’ютером, спирається на шлях від світла до камери, або навпаки. Роблячи це фізично реалістичним чином, ми можемо легко моделювати оптичні ефекти, такі як каустика або відображення світла іншою поверхнею в сцені. Підводячи підсумок, важливо пам’ятати, що процедуру рендеринга можна розглядати як два окремі процеси. Перший крок

визначає, чи видно точку на певному пікселі, а другий – тінь. Алгоритм елегантний і потужний, але змушує нас торгувати часом візуалізації для точності і навпаки.

4. ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

4.1. Методи оптимізації системи

Як було проаналізовано, трасування променів повністю функціональний алгоритм, але потребує значних обчислювальних витрат. Тому були реалізовані деякі ідеї для прискорення роботи трасування.

Найбільш очевидний спосіб пришвидшення роботи трасування променів полягає в тому, щоб трасувати декілька променів одночасно. Оскільки кожен промінь, який виходить з камери, незалежний від усіх інших, та більшість структур потрібні лише для перегляду, можна трасувати по одному променю на кожне ядро центрального процесора без яких-небудь складностей через проблем з синхронізацією.

Паралелізація. Насправді, алгоритм трасування променів вважається надзвичайно паралельним, тому що саме їх суть дозволяє дуже просто їх розпаралелювати.

Кешування даних. Зазвичай, трасування променів більше всього часу витрачає на обрахунок значень для променів сфери, але деякі значення розташування сфери, після їх виявлення, залишаються незмінними. Можна їх обрахувати один раз під час завантаження сцени та зберігати їх в самих сферах.

Оптимізація затінення. Якщо точка об’єкту в тіні знаходиться відносно джерела світла, тому що на її шляху знаходиться інший об’єкт, то ми отримуємо велику ймовірність того, що сусідня з нею точка, через той самий об’єкт, також знаходиться в тіні відносно джерела світла (Рис 4.1). [15]

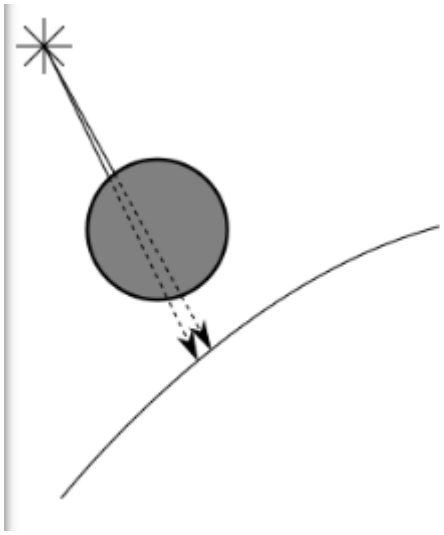

Рис. 4.1 – дві затіненні точки відносно одного джерела світла та об’єкту затінення [15]

Тобто коли ми шукаємо об’єкти між джерелом світла та заданою точкою, ми повинні спочатку перевірити, чи не накладається на поточну точку тінь від найближчого до неї об’єкту, що наклав тінь на попередню точку, відносно того ж самого джерела світла. У випадку, коли це так, ми можемо завершити, а якщо ні, треба продовжити звичайним методом перевіряти інші об’єкти.

Аналогічно, при обчисленнях перетину між променем світла та об’єктами нам не потрібно знати найближчий перетин, достатньо лише знати, що існує хоча б один перетин. Отже, використовується спеціальна версія ClosestIntersection, яка повертає результат, коли знаходиться перший перетин, і для цього треба повертати лише булеве значення.

Просторові структури. Обрахування перетину кожного променя з кожною сферою досить велика втрата часу. Існує велика кількість структур, яка дозволяє легко відкидати цілі групи об’єктів, що не потребують обчислень перетинів.

Уявімо, що є декілька сфер, які знаходяться досить близько один до одного. Можна обрахувати центр та радіус найменшої сфери, яка містить в

собі всі ці сфери. Якщо промінь не перетинає цю граничну сферу, то можна бути впевненим, що він не перетинає сфери, які містяться в головній. Це можна зробити за допомогою одної перевірки перетину, але якщо промінь перетинає сферу, то все одно треба перевіряти наступні сфери на перетин з ним.

Субдискретизація. Існує доволі простий спосіб зробити трасування променів в N разів швидше, тобто обраховувати в N разів менше пікселів.

Уявімо, що трасуються промені для пікселів (10, 100) та (12, 100), та вони падають на один об’єкт. Можна логічно припустити, що промінь для пікселя (11, 100) також буде припадати на той же об’єкт, пропустити початковий пошук перетинів з усією сценою, та перейти до обрахування кольору в цій точці. [12]

Якщо зробити так в горизонтальному та вертикальному напрямках, то можна виконувати максимум на 75% менше початкових обрахувань перетинів променів зі сценою.

Але в даному методі оптимізації існує один суттєвий недолік. Можна дуже просто пропустити дуже тонкий об’єкт, та результати використання оптимізації не будуть схожі на ті, що отримуються без неї. Головна ідея полягає в тому, щоб здогадатися, коли слід використовувати цей метод правильно, для отримання задовільних результатів.

Суперсемпінг. Суперсемпінг – це протилежність методу субдискретизації, якщо ми віддаємо перевагу точності замість шкидкості. Уявімо, що промені, які відповідають двом сусіднім пікселям, припадають на два різних об’єкта. Нам потрібно розмалювати кожен піксель у відповідний колір.

Кожен промінь повинен задавати колір для кожного квадрату сітки, через яку ми дивимося. Використовуючи по одному променю на піксель,

ми умовно вирішуємо, що колір променя світла, який проходить через середину квадрату, визначає цілий квадрат, але це не завжди так.

Для вирішення цієї проблеми можна трасувати декілька променів на піксель – 5, 10, 17, і так далі, а потім брати середнє значення, для отримання кольору пікселя. Це призводить до того, що трасування променів становиться повільніше в 5, 10 або 17 разів, завдяки тій же самій причині, по якій субдискретизація робить його в N разів швидше. Але існує компроміс. Можна припустити, що властивості об'єкта на протязі його поверхні змінюються плавно, тобто трасування 5 променів на піксель, які припадають на один об'єкт в сусідніх точках, не дуже сильно поліпшить вид сцени. Тому можна почати з одного променя на піксель та порівнювати сусідні промені: якщо вони припадають на інші об'єкти, або їх колір відрізняється більше ніж на порогове значення, то застосовуємо до них підрозділ пікселів. [13]

4.2. Методи тестування

Для базового тестування системи побудови зображень 3D моделей з використанням технології відстеження зворотної траєкторії променя було обрано 3 початкових сцени. Сцена 1 має всього лише декілька полігонів(Рис 4.2), друга має об'єкти з низькими багатокутниками(Рис 4.3), що розподілені в сцені, та в 3 сцені присутні високі і низькі багатокутні об'єкти(Рис 4.4). Всі тестування розробленої системи відбувались на комп'ютері за наступними технічними характеристиками:

- Процесор Intel Core i5-2450M, 2 ядра, 2.5 Ghz;
- Оперативна пам'ять DDR 3, 4 gb;
- Відеокарта GeForce 960;

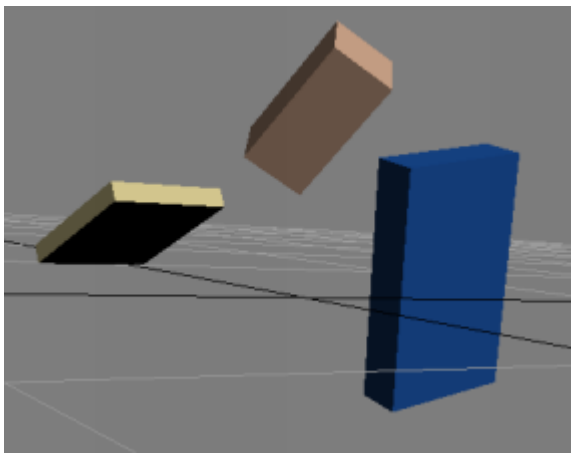

Рис. 4.2 – сцена 1

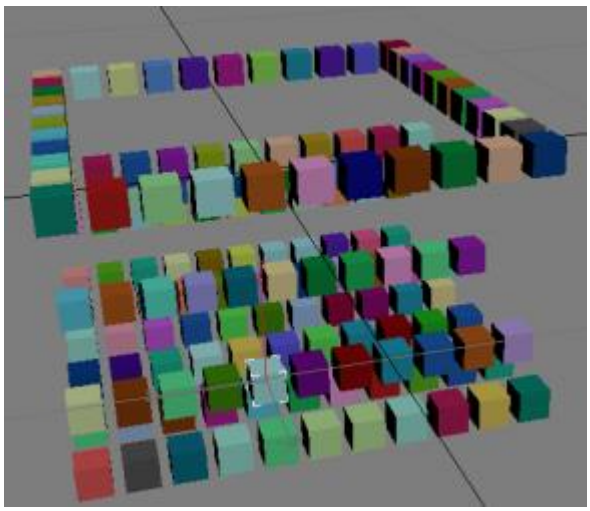

Рис. 4.3 - сцена 2

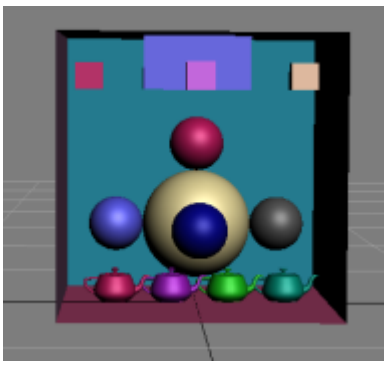

Рис. 4.4 – сцена 3

Тепер, для обраних сцен можна застосувати технологію трасування променів, та побачити результат відображення 3D моделей в сцені за допомогою реалістичних оптичних ефектів(Рис 4.5-4.6).

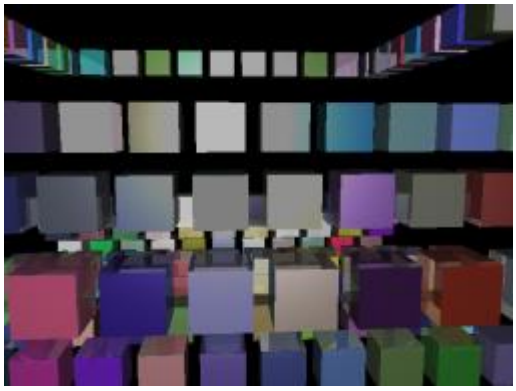

Рис. 4.5 – сцена 2 з застосуванням технології зворотного відстеження променя

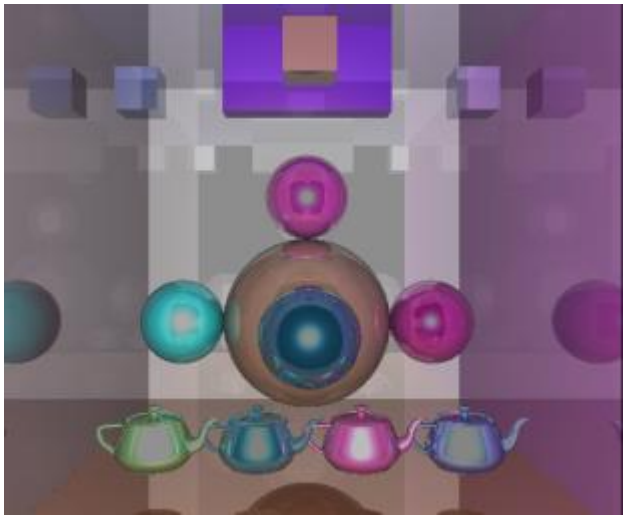

Рис. 4.6 - сцена 3 з застосуванням технології зворотного відстеження променя

Отже, наступним чином треба протестувати швидкодію побудови зображень 3D об’єктів відносно обраних методів оптимізації системи обраних сцен (Рис 4.7).

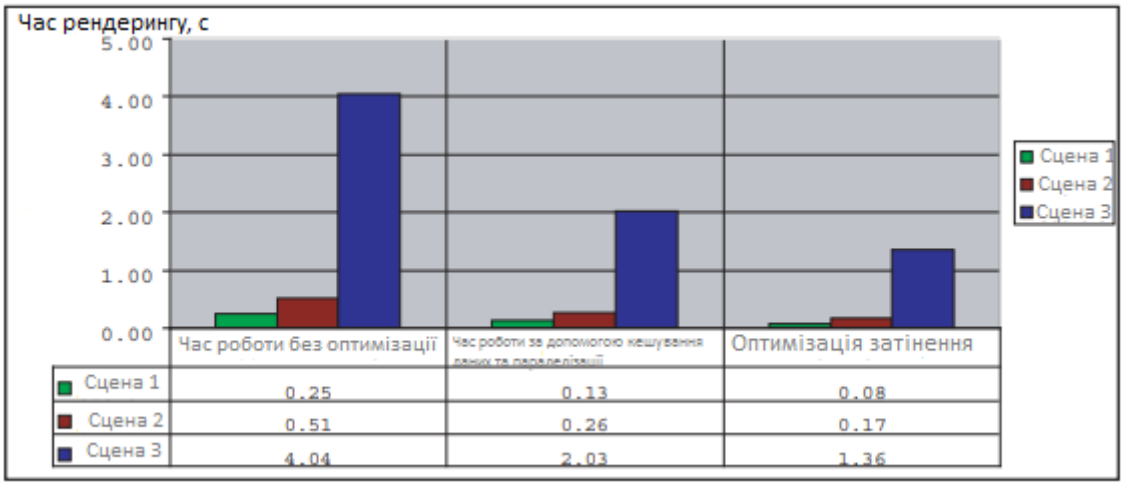

Рис. 4.7 – час рендерингу для різних методів оптимізації

4.3. Результати тестування

Для тестування результатів роботи розробленої системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя були побудовані зображення з наступними властивостями:

- Базова растеризація;
- Перетворення об’єктів за допомогою матриць (переміщення, обертання камери та джерела світла);
- Затінення;
- Білінійна інтерполяція;
- Відображення променів;
- Навколишнє освітлення сцени
- Відображення;
- Заломлення світла;

Отже, наступні приклади демонструють функціональну роботу системи.

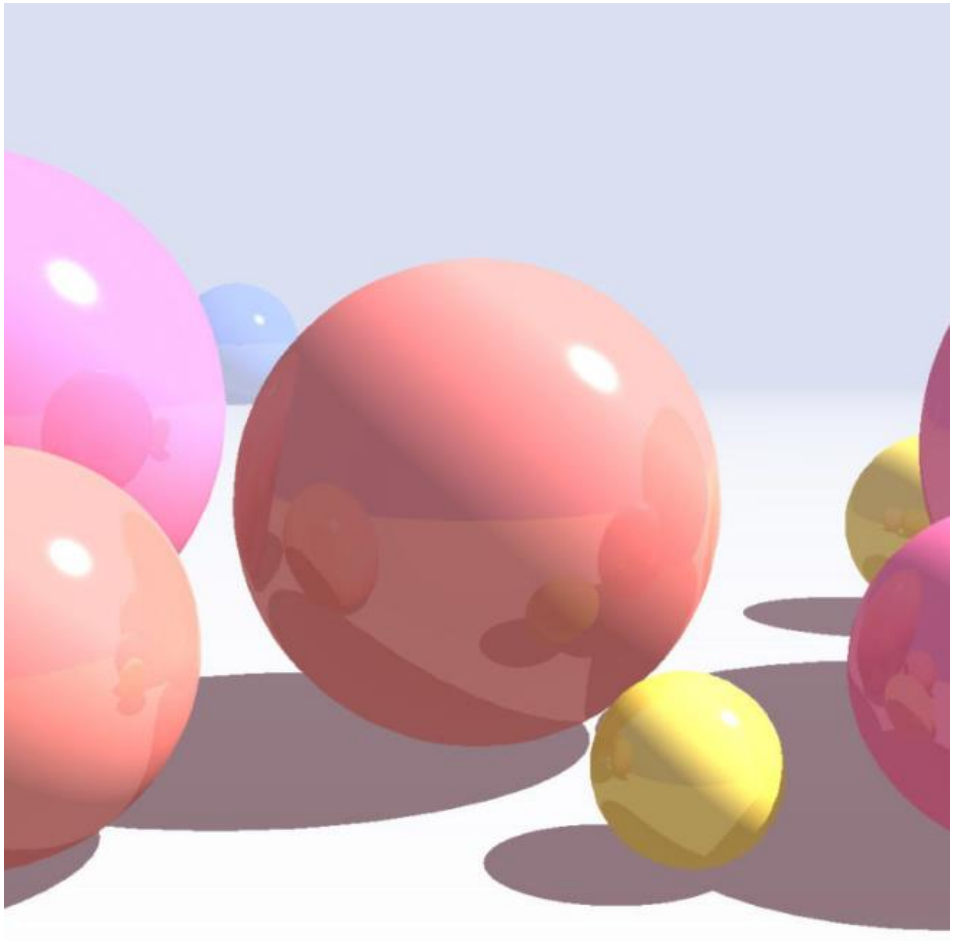

Рис. 4.8 – демонстрація відображення променів між сферичними об’єктами

Рис. 4.9 – чорно-білий промінь простежується на прикладі автомобіля

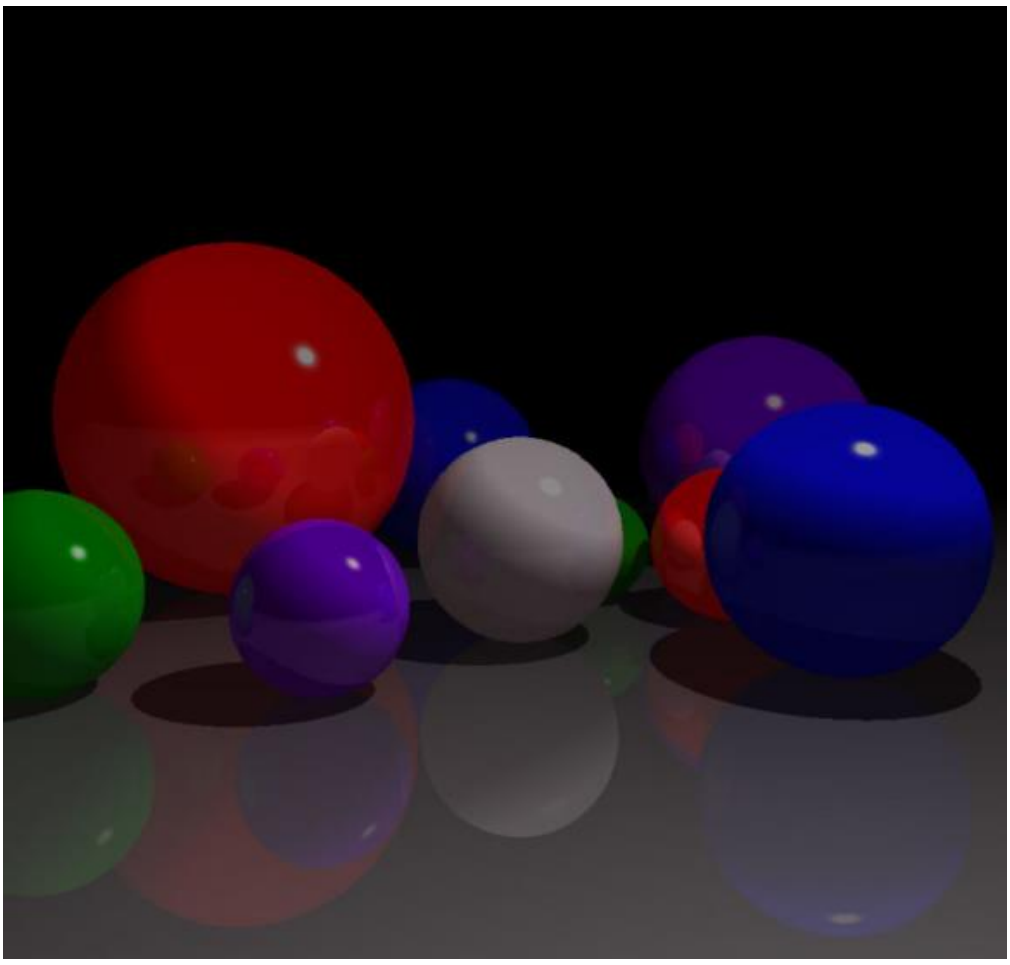

Рис. 4.10 - демонстрація відображення променів між сферичними об’єктами

Рис. 4.11 – демонстрація рендерингу об’єктів, таких як циліндр, куб, сфера, капсула, конус, чайник

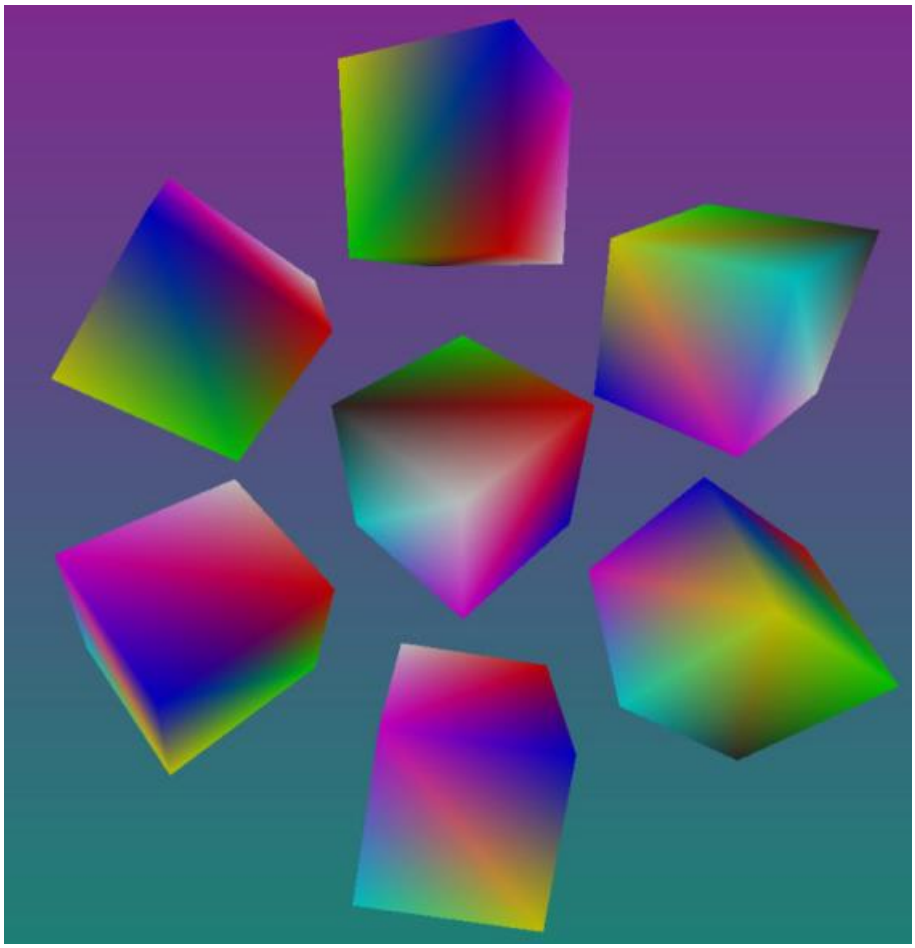

Рис. 4.12 – візуалізація з 7 кубів, що демонструють білінійну інтерполяцію

4.4. Висновки

В даному розділі були обгрунтовані методи оптимізації системи, такі як: кешування даних, розпаралелювання, оптимізація затінення, субдискретизація та суперсемпінг. Продемонстровано на малюнку функціонування методів оптимізації та доведена їх коректність. Крім цього, візуально показана робото-спроможність та фото-реалістичність системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя на багатьох прикладах. Також було протестована система на

слабкіших комп’ютерах, і виявлено, що вона може не працювати коректо, у зв’язку з тим, що система потребує доволі потужну обчислювальну спроможність.

ВИСНОВКИ

В дипломній роботі була розроблена система побудови зображеня 3D моделей з відстеженням зворотної траєкторії променя.

Основні результати, отримані в роботі, можна сформулювати наступним чином.

1. Виконаний аналіз існуючих альтернативних методів побудови зображення, їх переваг та недоліків.
2. Детально розібрана техніка візуалізації зворотного відстеження променя, описана функція трасування та проаналізовані етапи обчислення фото-реалістичних 3D зображень.
3. Забезпечена фото-реалістичність системи за допомогою дослідження таких основних оптичних явищ, як затінення, заломлення світла та відображення.
4. Система оптимізована, та може забезпечувати швидкодію на 60% більше, ніж без запропонованих методів оптимізації. Реалізовано за допомогою наступних методів: кешування даних, розпаралелювання, оптимізація затінення, просторові структури, субдискретизація та суперсемпінг.
5. Забезпечена кросплатформність системи, тобто може функціонувати як на Windows, так і на Linux.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Тернер Уіттіс. «Покращена модель освітлення для затіненого дисплея», 1980 [Text].

2. Роберт Кук, Томас Портер, Лорен Карпентер. «Розподілене трасування променів», 1984 [Text].

3. J. Avro, and D. Kirk. A survey of ray tracing acceleration techniques. In An Introduction to Ray Tracing, A. Glassner, Ed., pages 201–262. Academic Press, San Diego, CA, 1989 [Text].

4. Артур Аппель. «Деякі методики відтінку твердих тіл», 1969 [Text].

5. Кевін Сафферн. «Ray Tracing from the Ground Up», 2007 [Text].

6. HEKPIKS.ORG [Electronic resource] https://helpiks.org/3-2158.html - Last access: 2019.

7. NVidia Corporation, NVIDIA GPU Programming Guide Version 2.2.0, 2004 [Text].

8. John Amanatides, Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing, 1987 [Text].

9. Michael McCool. http://libsh.sourceforge.net/, 2004 [Electronic resource].

10. Интерактивная трассировка лучей с использованием SIMD инструкций INTEL [Electronic resource]. / Intel, 2009. - Режим доступа: http://software.intel.com/ru-ru/articles/ interactive-ray-tracing.

11. Yunfan Zhang. FPGA Ray Tracer: [Electronic resource]. – Режим доступа: http://www.eeweb. com/project/yunfan_zhang/fpga-ray-tracer.

12. Юніс Мохаммад. Залучення блочної міжпіксельної інтерполяціх для прискорення алгоритму трасування променів / Мохаммад Юніс, Р.В. Мальчева, С.О. Ковалев // Машинобудування та техносфера XXI століття. / Сбірник XVIII міжнародної научної конференції. - Донецьк: ДонНТУ, 2011. - Т.4. - С.189-191 [Text].

13. Юніс Мохаммад. Залучення блочної міжпіксельної інтерполяціх для прискорення алгоритму трасування променів / Мохаммад Юніс, Р.В. Мальчева // Донецк-Таганрог: ДонНТУ, 2011. – Т.2. – С.72-74 [Text].

14. Jorg Schmittler, Daniel Pohl, Tim Dahmen, Christian Vogelgesang, and Philipp ¨ Slusallek. Realtime Ray Tracing for Current and Future Games, 2004 [Text].

15. https://habr.com/ru/post/187720/ [Electronic resource].